Last time…

We discussed

- Analysis of ArrayLists vs arrays
- Garbage Collection
- Input file redirection
- Recursion

Quiz Wednesday!

- Quizzes are good prep for the final exam

GO OVER QUIZ #5 in class

CS112 –
Java
Programming

Fall 2022

Recursion

Recursion

Example: Fibonacci sequence

1, 1, 2, 3, 5, 8, 13, 21, 34, …

Recursion is a way to think about solving some problems. If it can be used, solution can be elegant. But obviously a bit tricky to think about. You will be getting some experience thinking about solving problems in recursive ways during the rest of the semester.
DON'T OVERUSE! Is useful "mind map" and is definitely used, but is tricky enough that it should be used only when needed.
Like inheritance.

Recursive Programming

A recursive method is a method that invokes itself

A recursive method must be structured to handle both the base case and the recursive case

Each call to the method sets up a new "execution environment", with a new set of method parameters and local variables

As with any method call, when the method completes, control returns to the method that invoked it (which may be an earlier invocation of itself)

Sum of 1 to N

Consider the problem of computing the sum of all the numbers between 1 and any positive integer N

This problem can be recursively defined as:

$$\sum_{i=1}^{N} i = N + \sum_{i=1}^{N-1} i = N + N - 1 + \sum_{i=1}^{N-2} i$$

$$= N + N - 1 + N - 2 + \sum_{i=1}^{N-3} i$$

$$\vdots$$

$$= N + N - 1 + N - 2 + \cdots + 2 + 1$$

Sum of 1 to N

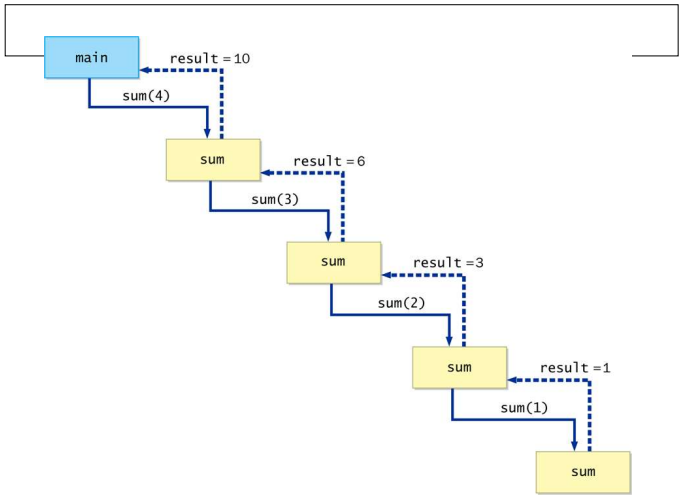
The summation could be implemented recursively as follows:

```
// This method returns the sum of 1 to num
public int sum(int num)
{
   int result;

   if (num == 1)
      result = 1;
   else
      result = num + sum(n-1);

   return result;
}
```

Sum of 1 to N



We have multiple “calls” to sum() all operating in parallel, waiting for results from the “lower” sum() call.

Recursive Programming

Note that just because we can use recursion to solve a problem, doesn't mean we should

We usually would not use recursion to solve the summation problem, because the iterative version is easier to understand

However, for some problems, recursion provides an elegant solution, often cleaner than an iterative version

You must carefully decide whether recursion is the correct technique for any problem

Recursion Overuse

Factorial

Fibonacci

Solve each of these with a loop!

When should we use Recursion?

There are some problems that are easiest to solve by solving the same problem on a slightly simpler/smaller dataset

Often involves processing linked data structures

Like our lists
THINK ABOUT THE TERMINATION CONDITION!

Towers of Hanoi

The *Towers of Hanoi* is a puzzle made up of three vertical pegs and several disks that slide onto the pegs

The disks are of varying size, initially placed on one peg with the largest disk on the bottom with increasingly smaller ones on top

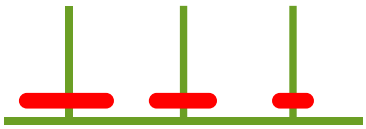
The goal is to move all of the disks from one peg to another under the following rules:

- Move only one disk at a time
- A larger disk cannot be put on top of a smaller one

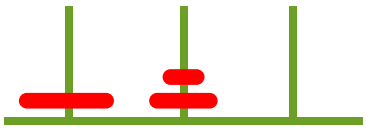
Towers of Hanoi

Original Configuration

Move 1

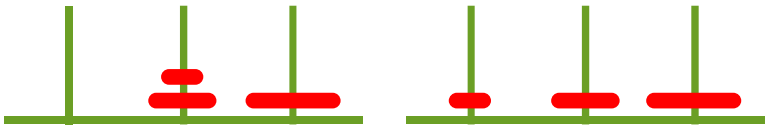


Move 2



Move 3

Towers of Hanoi



Move 4

Move 5

Move 6

Move 7 (done)

Towers of Hanoi

An iterative solution to the Towers of Hanoi is quite complex

A recursive solution is much shorter and more elegant

See Eclipse! Towers.java
SHOW THE CODE -- RUN IT!

Recursion Practice

We will use recursion a lot for the rest of the course

- Today's homework
- Next several lectures
- Second big project

Interlude

Debugging practice – Lab17 using Eclipse

Tree Data Structures

Trees

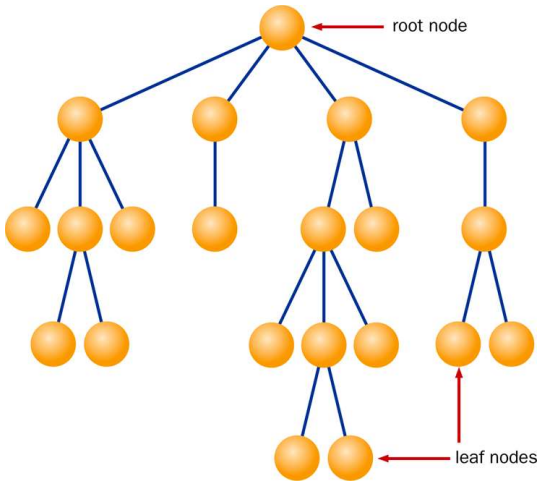
A *tree* is a non-linear data structure that consists of a *root node* and potentially many levels of additional nodes that form a hierarchy

Nodes that have no children are called *leaf nodes*

Nodes except for the root and leaf nodes are called *internal nodes*

In a general tree, each node can have many child nodes

A General Tree



Binary Trees

In a *binary tree*, each node can have no more than two child nodes

Trees are typically are represented using references as links to child nodes

For binary trees, this requires storing only two links per node to the left and right child

For nonbinary trees, use an ArrayList of references.

Designing a class Tree

```
class TreeNode {
 String nodeValue;
 TreeNode left;
 TreeNode right;
 TreeNode(String s) {
      nodeValue = s; left = null; right = null;
 }
}
…
TreeNode root = new TreeNode("I am Root");
root.left = new TreeNode("I am left");
root.right = new TreeNode("I am right");
```

When would we use a Tree data structure?

Representing data that has hierarchy

- Animal
  - Reptile
    - Snake
    - Lizard
  - Bird
    - Turkey
    - Sparrow
  - Mammal
    - Horse
    - Cow
    - Lion

When would we use a Tree data structure?

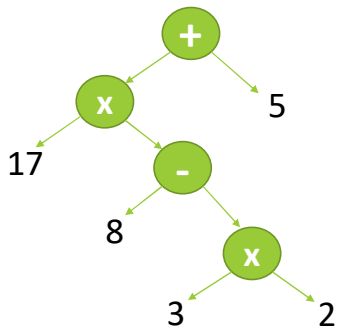
Computer filesystem with directories and files

- Root
  - /usr
    - /usr/bin
      - /usr/bin/mv
      - /usr/bin/ls
  - /home
    - /home/phaskell
    - /home/stephcurry
  - /dev
    - /dev/printer
    - /dev/keybd

When would we use a Tree data structure?

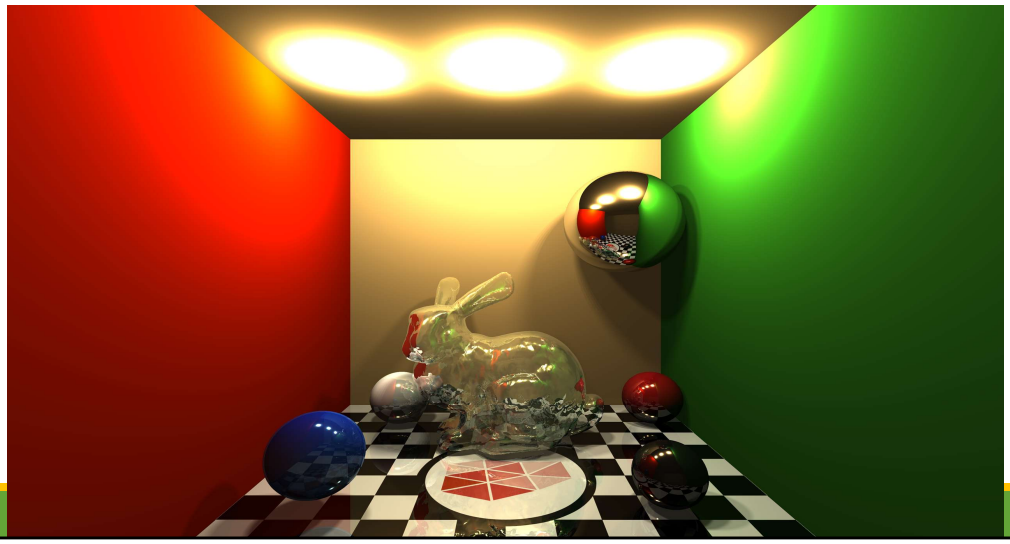
Expression parsing

```
x = 17 * (8 – 3*2) + 5;
```



When would we use a Tree data structure?

Computer graphics, video rendering



When would we use a Tree data structure?

Some specialized data processing algorithms

- HeapSort
- Huffman Coding
- Computer Chess
- etc

Our second project will work with Huffman Coding

Trees and Recursion

Trees and Recursion

Working with Trees is often easiest with recursion

- Apply same operation to tree child branches as to top
- Every node is the top of its own subtree

```
TreeNode root;
…
void CountLeafNodes(TreeNode treeRef) {
 if (left == null && right == null) { return 1; }
 return CountLeafNodes(left) + CountLeafNodes(right);
}
```

Draw a tree
Walk through it!
Really short, elegant code. Hard part is learning how to “express a problem recursively”

Trees and Recursion

```
TreeNode root;
…
boolean FindValue(TreeNode treeRef, String toFind) {
 if (left == null && right == null) {
     return value.equals(toFind);
 }

 boolean retval = FindValue(left, toFind) || FindValue(right, toFind);
 return retval;
}
```

Draw a tree
Walk through it SLOWLY. Show all calls to FindValue()

Trees and Recursion

```
TreeNode root;
…
boolean FindValue(TreeNode treeRef, String toFind) {
 if (left == null && right == null) {
     return value.equals(toFind);
 }

 boolean retval = FindValue(left, toFind) || FindValue(right, toFind);
 retval |= value.equals(toFind); // if non-leaf nodes have values
 return retval;
}
```

Explain text in red